(2)

CDRL No. 0002AC-2

Description of CE Technology For The Manufacturing Optimization (MO) System

Linda J. Lapointe
Robert V.E. Bryant

Raytheon Company

1992

DARPA
Defense Advanced Research
Projects Agency

CDRL No. 0002AC-2

Description of CE Technology For The Manufacturing Optimization (MO) System

Prepared by

Linda J. Lapointe

Robert V.E. Bryant

Raytheon Company

Missile Systems Laboratories

Tewksbury, MA 01876

March 1992

ARPA Order No. 8363/02

Contract MDA972-92-C-0020

Prepared for

DARPA

Defense Advanced Research Projects Agency

Contracts Management Office

Arlington, VA 22203-1714

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave Blank)*

2. REPORT DATE
19 March 92

3. REPORT TYPE AND DATES COVERED
Final

4. TITLE AND SUBTITLE
Description of CE Technology For The Manufacturing Optimization (MO) System

5. FUNDING NUMBERS
(C) MDA972-92-C-0020

6. AUTHOR(S)
Linda J. Lapointe and Robert V. E. Bryant

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Raytheon Company
50 Apple Hill Drive
Missile Systems Laboratories
Tewksbury, MA 01876

8. PERFORMING ORGANIZATION REPORT NUMBER
CDRL No. 0002AC-2

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency (DARPA)
Contracts Management Office
Arlington, VA 22203-1714

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This is the Description of the Concurrent Engineering (CE) Technology For the Manufacturing Optimization (MO) System document. The purpose of the Manufacturing Optimization (MO) system is to enable each manufacturing specialist to participate in the product/process development activity concurrently. The system will consist of a set of tools to model the manufacturing processes and centralize the various process tradeoffs. The subject of this report is involved in "user hardening" of existing concurrent engineering technology by applying existing DICE Tools to other technology areas, in this case design for manufacturing and assembly (DFMA). The DICE tools and methods are evolving so evaluation of the existing DICE tools must be performed, and the tools that support the DFMA environment will be considered for incorporation into the MO system. This report identifies the methods, assumptions, and procedures taken to evaluate the DICE tools (ROSE, CMS, PCB, CM, and RM) being considered for incorporation into the MO system, as well as, the results, conclusions, and recommendations produced by the evaluation.

14. SUBJECT TERMS
MO, CE Technology, ROSE, RM, CMS, PCB, and CM.

15. NUMBER OF PAGES
52

16. PRICE CODE

17. SECURITY CLASSIFICATION OF THIS REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std Z39-18
298-102

Contents

Tables

Figures

1. Summary

This is the Description of the Concurrent Engineering (CE) Technology for the Manufacturing Optimization (MO) System. The purpose of the Manufacturing Optimization (MO) system is to enable each manufacturing specialist to participate in the product/process development activity concurrently. The system will consist of a set of tools to model the manufacturing processes and centralize the various process tradeoffs. This report addresses "user hardening" of existing concurrent engineering technology by applying existing DICE tools to other technology areas, in this case Design for Manufacturing and Assembly (DFMA). The DICE tools proposed for use under the MO program included ROSE Database Management System (DBMS), Constraints Management System (CMS), and the Requirements Manager (RM). The Project Coordination Board (PCB) and Communications Manager (CM) were added to the scope of the evaluation because the September release of the CMS will be integrated with the PCB/CM.

Raytheon proposed to use ROSE to store and manage the process models and analysis data for the MO system. As part of the evaluation process of ROSE, we developed a set of test cases to demonstrate the applicability of the ROSE DBMS to adequately model the manufacturing process models and analysis data. Based on the results of our sample test case, we believe that the ROSE DBMS is able to handle the various types of manufacturing process model and analysis data required in the MO environment. We recommended the continued use of the ROSE DBMS as the repository for all the process data files for the manufacturing processes and operations, as well as, the various analysis results for the MO system. To better understand the more advanced features of ROSE, attending the three day training session conducted by Step Tools Inc. is recommended.

Raytheon proposed to use the CMS to provide a mechanism for constraining the product design to process capabilities. Since the pilot site version of the CMS software was unavailable for testing, Raytheon planned a trip to CERC for a demonstration of the CMS and a meeting with the developers. Based on the CERC visit, it was determined that the future versions of CMS will support the product/process constraint mechanism proposed in MO. Raytheon plans on installing the June release of the CMS. Raytheon will continue to develop the process selection functionality and will evaluate the CMS when it is available. If the CMS is not used, the implemented MO process selection functionality will include mechanisms such that it can be integrated into the product team environment as a module.

The types of constraints that the next CMS version will be able to handle are the bidirectional constraint represented by algebraic equations linked to Mathematica™, and the blackbox constraint (unidirectional) which

receive input and produce output through the evaluation of an external analysis program. Many of the constraints that the MO system would require come in the form of conditional and logical constraints. In order to integrate the first release of the CMS, we would have to utilize the blackbox constraint by writing our own external analysis program that would take our constraint format as input and return the result as output.

Raytheon reviewed the PCB and CM which will be integrated with the September version of the CMS software. Based on the results of the hands on PCB/CM training class, Raytheon saw a potential to use the tools as a front-end for MO data model entry due to its hierarchical product structure display. We have reservations about the maturity of the system, and its interfaces with other systems (i.e. ROSE). The recommended plan is to get the PCB/CM software installed at Raytheon so that we can perform an internal evaluation using sample MO data. During our evaluation, we would be able to determine if the PCB/CM would add any value to the MO environment as an extension to the CMS or as a front-end for MO process data model entry.

The Requirements Manager (RM) is a system designed to manage product requirements, specifications and corporate policies to support concurrent engineering. Within the MO program, Raytheon planned to use the RM for manufacturability/producibility guidelines. Version 2.0 of the RM was installed at CERC, but is not supported. Raytheon decided to wait for the upcoming release, Version 3.0, and not to install V2.0. Raytheon attended a demonstration of the RM V3.0 at the DICE Phase IV Kick-off meeting. According to announced plans, the RM V3.0 will be available for beta test at the end of March and released for production use at the end of June. Raytheon has maintained a dialogue with CIMFLEX and has expressed its interest in being a beta test site. The purpose of integrating the manufacturing guideline functionality into the RM is to give the "top level" product development team insight into manufacturing requirements apart from MO analyses. Raytheon will continue to develop the MO guideline functionality and will evaluate the RM when it is available. If the RM is not used, the implemented MO guideline functionality will include mechanisms such that it can be integrated into the product team environment as a module.

2. Introduction

The Manufacturing Optimization (MO) system is a conceptual refinement to the original DICE virtual tiger team concept. This refinement is to have a two level approach with a product virtual team having a global view supported by information supplied by lower level "specialized" process virtual teams. The purpose of the Manufacturing Optimization (MO) system is to enable each manufacturing specialist to participate in the product/process development activity concurrently. The system will consist of a set of tools to model the manufacturing processes and centralize the various process tradeoffs. The primary mission of the MO system is in researching and developing a "generalized" design for manufacturing and assembly (DFMA) environment capable of modelling diverse manufacturing processes. The system will consist of five software modules: process analysis, yield/rework modeler, cost estimator, guidelines, and manufacturing advisor. Refer to reference 2 for more details on the MO system. The secondary mission, and subject of this report, is the "user hardening" of existing concurrent engineering technology by applying existing DICE Tools to DFMA.

During the proposal stage of MO, Raytheon did a preliminary evaluation of the concurrent engineering tools previously developed under the DICE program. The evaluation consisted of visits to both CERC and Rensselaer Polytechnic Institute (RPI) to attend demonstrations and presentations of DICE tools. Available technical papers for each of the DICE tools were studied by the proposal team. Each of the tools demonstrated at the CERC appeared to be somewhere between a prototype to a released product. Each DICE tool had been developed and/or demonstrated either stand-alone or partially combined on a single technology (e.g. turbine blade). Based on this preliminary evaluation, the ROSE Database Management System (DBMS), Constraint Management System (CMS), and Requirements Manager (RM) were identified as candidates for incorporation into the MO system. Raytheon also planned on using the EDEX Express editor to code and edit Express schemas. No evaluation of the EDEX editor was planned or conducted as part of this report.

The CMS is scheduled to be integrated with the Project Coordination Board (PCB) and Communications Manager (CM) by the end of September so Raytheon expanded the scope of the evaluation to include the PCB and CM due to this close coupling among the PCB, CM and CMS.

Raytheon proposed to use ROSE to store and manage the manufacturing process models, CMS to constrain the product design to process capabilities, and RM to manage manufacturing guidelines. The proposed MO architecture is shown in Figure 2-1.

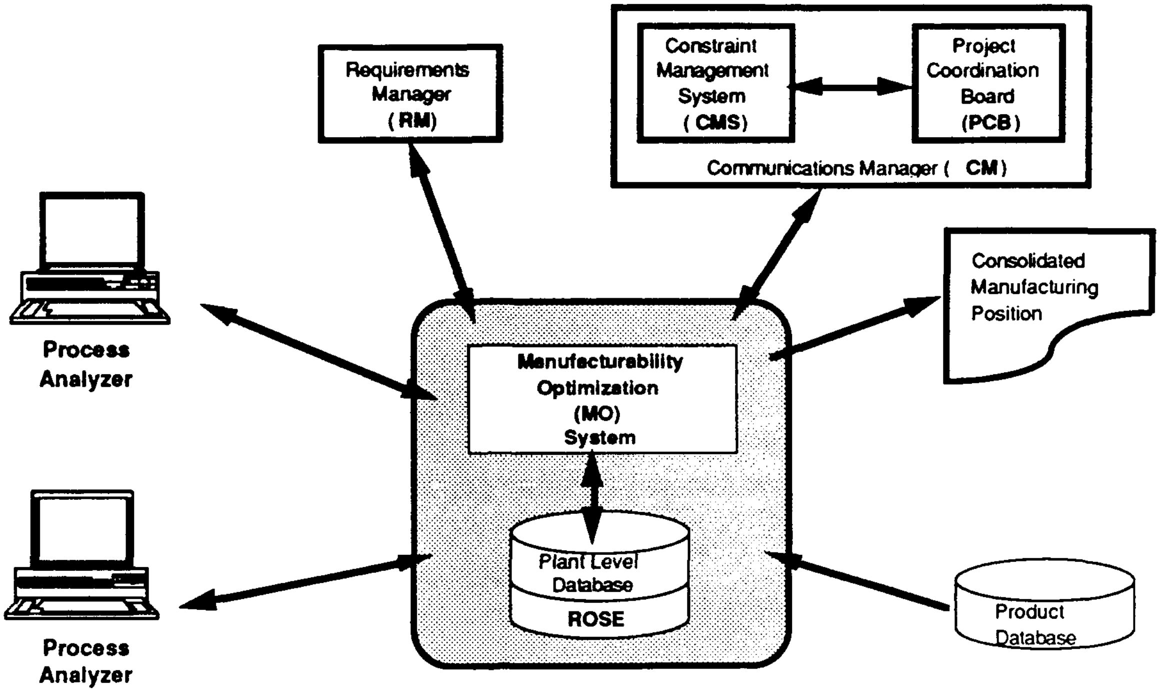

Figure 2-1. Proposed DICE MO Architecture

The purpose of the evaluation described herein was to determine the maturity of each of the DICE tools and confirm their applicability to MO. Where the tools are not as mature as expected, Raytheon has proposed alternative approaches. The basic evaluation plan was to install the latest available version of each of the tools in our own environment and test the tools by executing a sample test set of data that was representative of the basic MO functionality. The remainder of this report is devoted to the methods, assumptions, and procedures, as well as, the results, conclusions, and recommendations of the evaluation of each of the tools identified.

3. ROSE - Rensselaer Object System For Engineering

3.1 Methods, Assumptions, and Procedures

3.1.1 Overview

ROSE is an object-oriented database management system that has been developed for engineering applications and enhanced to support the DICE program. The ROSE Database Management System is a database that supports concurrency using a data model that allows the differences between two design versions to be computed as a delta file. ROSE will be used to store and manage the data files for the manufacturing processes and operations, as well as, the various analysis results. Its proposed use in the MO system is to store the process routing sequence which will enable us to do comparisons of process trade-offs via the delta files. The process database consists of the process selection knowledge base, process/operation data, yield/rework data, guidelines, and recommendations.

The EDEX Editor is an editor that has been developed to support EXPRESS schema creation and manipulation for ROSE. EDEX will be used to code and edit the manufacturing process model schema(s).

3.1.2 Proposed Use In MO

The MO system will consist of five software modules: process analysis, yield/rework modeler, cost estimator, guidelines, and manufacturing advisor. The process analysis module performs the initial analysis on the design to determine the manufacturing process required to produce the part. This module will take a knowledge based approach that will compare design features and attributes against process capabilities to determine a part's process sequence. The actual process will be modelled as a hybrid decision tree with rules attached to each node of the tree. The decision tree informs the user of the basic flow of the overall process while letting the user plan at various levels of abstraction. These levels include the process, operation, and operational step. The process is an organized group of manufacturing operations sharing characteristics, the operation is a common unit of work that is performed on the part, and the operational step is an elemental unit of work within an operation. Before an analysis can be run on a design, the process database must be populated. The process database consists of the process selection knowledge base, process/operation data, yield/rework data, cost data, guidelines, and recommendations. We propose to have the ROSE DBMS manage all the process model data.

Once the process database is populated, evaluation of the design is possible. The process analysis module executes the process knowledge base to select a process and establish the list of operations that will produce the part. After the operation sequence is determined, each applicable operation is evaluated to calculate yield and rework values. These values are established by the relationship between the design and the operation. The cost is calculated based on the labor standards associated with each operation which are factored by the yield and rework values. This method accounts for the cost of scrap and rework. We plan to store the analysis results in the ROSE DBMS.

Once each participating team member has run an analysis within the MO system, the data is consolidated within ROSE through the manufacturing advisor module. The advisor organizes the data by summarizing processes, guideline violations, and recommendations. From this summary, the process team members can identify major costs drivers and prioritize change recommendations. These recommendations could then be sent to the product team. The process team could also elect to re-run each analysis using the recommended changes to compare the costs of the proposed design with the original design.

3.1.3 Evaluation Plan

Raytheon developed a set of test cases to demonstrate the applicability of the ROSE DBMS to adequately model the manufacturing process models and analysis data. We modelled the three levels of the decision tree abstraction in the EXPRESS information modelling language so that we could create (populate) a sample manufacturing process model database. We used the actual process model we stored in ROSE to select two different process plan sequences and store those results in another ROSE database. Finally, we read in the list of process plan results so that we could evaluate the process of capturing results from a ROSE database for the manufacturing advisor module to perform comparisons and summations. Detailed below are the actual steps that we performed to evaluated the ROSE DBMS.

Step 1: Installed a copy of the Step Tool Kit (ROSE) Release 1 that was sent to us from Step Tools Inc.

Step 2: Performed the eight tutorial lessons provided in the ROSE Tutorial Manual (Reference 6) to become familiar with ROSE functionality and features.

Step 3: Modelled a small sample of the process model data (process decision, operation, operational step, and a process plan/sequence) using the EXPRESS information modelling language.

Step 4: Generated the C++ classes for each entity in the EXPRESS schema using the "express2c++" tool.

Step 5: Compiled the generated C++ classes.

Step 6: Wrote, compiled, and ran a C++ program (ROSE Application) which created instances (STEP objects) of the generated C++ classes from an input ASCII data file.

Step 7: Created the input ASCII file for the above C++ program.

Step 8: Wrote, compiled, and ran a C++ program (ROSE Application) that read in the instantiated process model ROSE database and then selected, created, and stored two process plan/sequence objects.

Step 9: Wrote, compiled, and ran a C++ program (ROSE Application) that read the process plan database (output of the above program) into memory.

Step 10: Coordinated with the GE developers to acquire a copy of the latest version of the EDEX editor for us to try during our EXPRESS information modelling development efforts for the MO system.

3.2 Results and Discussion

We installed the Step Tool Kit that we received from Step Tools Inc. and performed the eight tutorial lessons provided in the ROSE Tutorial Manual (Reference 6). This provided us with an understanding of the basic ROSE functionality. After the initial learning curve, we developed a sample schema to test the applicability of the ROSE DBMS to adequately model the manufacturing process model and analysis data using the EXPRESS information modelling language (refer to Appendix I section 10). This schema modelled a small sample of the basic process model data types. Included are entities for a process decision, operation, operational step, and a process plan/sequence. We modelled the operation and operational step entities as subtypes of the process decision. The process plan/sequence was then modelled as a list of process decisions. Since we utilized inheritance when modelling the operation and operational step, a process plan/sequence would be able to contain a process decision, operation, and/or operational step. After developing the EXPRESS schema, we used the "express2c++" tool provided in the toolkit to generate the C++ classes for each of the entities in the schema. Refer to Appendix I section 20 for complete details on all the "express2c++" generated class code. Following is a diagram which depicts the process model express schema versus the corresponding "express2c++" generated classes.

```
SCHEMA process_model;

  ENTITY Decision;
    name: STRING;
    parent: STRING;
    rules: LIST [0:?] OF STRING;
    END_ENTITY;

  ENTITY Operation
    SUBTYPE OF (Decision);
    op_desc: STRING;
    bid_code: INTEGER;
    END_ENTITY;

  ENTITY Step
    SUBTYPE OF (Decision);
    step_no: INTEGER;
    step_desc: STRING;
    cutting_tool: STRING;
    no_passes: INTEGER;
    step_time: REAL;
    END_ENTITY;

  ENTITY Process;
    sequence: LIST [0:?] OF Decision;
    END_ENTITY;

END_SCHEMA;
```

express2c++

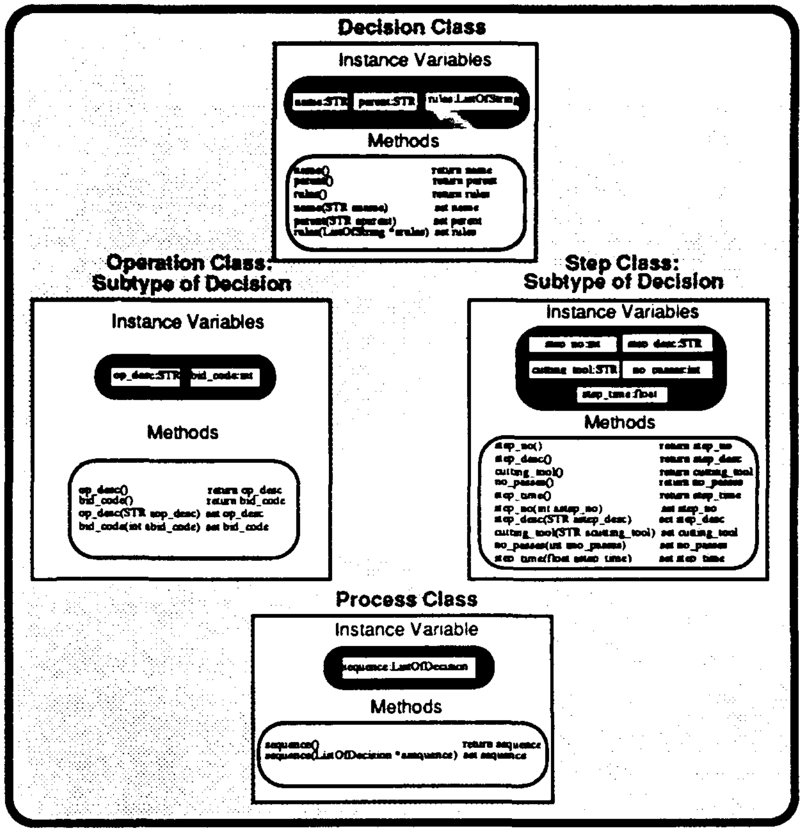

Figure 3-1. Process_Model EXPRESS Schema versus Generated C++ Classes

The next step involved the compilation of the generated C++ class code. When compiling the classes, the compiled process model schema was generated which represents the data structure definitions. Appendix I section 30 contains a print out of the compiled process model schema.

To test how ROSE could handle storage of a manufacturing process model database, we developed a ROSE Application (Appendix I section 40) that would use the generated C++ classes to create instances of the STEP objects by reading in sample data from an ASCII input file. The ASCII input file (refer to Appendix I section 50) contains a series of process decisions, operations, and operational steps. The program created the STEP objects that were represented in the input file and finally stored all the objects in the process model database as a List of Decisions. The output database from the execution of the ROSE Application can be found in Appendix I section 60.

A major part of what MO expects ROSE to handle is the storage of a particular process sequence which will be generated by the process analysis module inside the MO system. To test the ability of ROSE to handle this type of data, we developed a ROSE Application (Appendix I section 70) that read in the process model database that we previously generated, and then selected two sequences of decision objects based on their name.

We then created and stored these decision objects as two process objects. Refer to Appendix I section 80 for a complete print out of the output process plan/sequence database.

Finally, to test the ease in which we could read a complete list of process plan/sequence objects back into memory for the manufacturing advisor module to perform comparisons and summations, we developed a ROSE Application (Appendix I section 90) that simply read the previously generated process plan/sequence database into memory.

We contacted General Electric (GE) about obtaining a copy of the Editor for EXPRESS (EDEX) that was developed as an extension to the emacs editor which would help navigate through complex EXPRESS schemas. Raytheon has received documentation. GE will contact us in the near future to send us the EDEX editor. The plan is to install the EDEX editor on our system so that we can use it during our MO development efforts.

Learning the basics of the ROSE DBMS with the aid of the reference and tutorial manuals provided with the software (References 4 and 6) was straightforward. To better understand the more advanced features of ROSE, we plan on attending one of the training sessions conducted by Step Tools. Based on the results of our sample test case, we believe that the ROSE DBMS is able to handle the various types of manufacturing process model and analysis data required in the MO environment. Refer to Table 3-1 for a summary of the evaluation results of the system.

Table 3-1. Summary of the ROSE Evaluation Results

System Under Evaluation	MO Applicability	Maturity/Availability	Conclusion/ Recommendation
ROSE	Model, Store, and Manage Process Model and Analysis Data.	Stable. No difficulty with installation or evaluation procedures.	Plan on continuing integration efforts.
EDEX Editor	Code and edit MO Express Schemas.	No evaluation planned or conducted.	Plan to install.

4. CMS - Constraint Management System

4.1 Methods, Assumptions, and Procedures

4.1.1 Overview

The Constraint Management System allows the user to build constraints into the concurrent engineering process. The system tracks constraints and can notify, or launch an evaluation when constraints are not satisfied. The CMS will be used to manage the constraint placed on the design by the capabilities of the process required by the MO system.

The Project Coordination Board (PCB) is a system being developed to provide support for the coordination of the product development activities in a cooperative environment. The PCB provides common visibility and change notification through the common workspace, planning and scheduling of activities through the task structure, monitoring progress of product development through the product structure (i.e. constraints), and computer support for team structure through messages. This tool is currently integrated with the CM, and the CMS is scheduled to be integrated with the PCB by the second pilot site release.

The Communications Manager (CM) is a collection of modules that facilitates distributed computing in a heterogeneous network. It promotes the notion of a virtual network of resources which the project team members can exploit without any prior knowledge of the underlying physical network. The CM would be useful for those that would like to build transparent tools, virtual project networks, have access to remote tools, perform network tasks, perform message passing, and/or perform inter-process file transfers.

4.1.2 Proposed Use In MO

The rules that determine the selection (or evaluation) of operations within the MO system can be modelled as a set of constraints. If the constraints are satisfied then the operation can produce the part or feature (based on scope of rule set). If the rules or constraints of the operation are not satisfied then the operation is not valid for use. By interfacing with the constraint manager, these process/operation rules can monitor the design and be informed when the design has exceeded process capabilities. This can function by informing the team that the current design is limiting process alternatives or that the initial process evaluated for the design is no longer valid and a re-evaluation must be performed. In this scenario the re-evaluation will result in a process change.

For this function or set of functions, Raytheon planned two approaches. The first one would enable the manufacturing specialist to run the system and determine the constraint or requirement violations and interrelationships from the MO program. The second approach would integrate that piece of the analysis in another DICE tool so that the product team could be informed of the analysis immediately. The CMS, PCB, and the CM are the DICE tools considered for use in the second approach.

4.1.3 Evaluation Plan

The Constraint Management System went though initial development, and was loaded onto the CERC testbed and tested. From the results of the testing, the CMS was found to be unsuitable for pilot (beta) site installation. A future release of the CMS will be integrated with the PCB and CM.

Raytheon received papers on the CMS and read these as a comparison to requirements prior to our CERC visit. To conduct the evaluation of the CMS without benefit of site installed software, Raytheon planned a visit to CERC to attend a demonstration of the CMS and meet with the developers. The trip coincided with the CERC PCB/CM training session so Raytheon also attended the two day class to gain insight into the functionality of the PCB and CM since the CMS is scheduled to be integrated with the PCB/CM.

4.2 Results and Discussion

Prior to visiting CERC, we read two papers (Reference 3 and 5) on the CMS so that we could compare the requirements outlined in the paper with the scheduled first release requirements. On February 26th and 27th, Raytheon received a demonstration of the current version of the Constraint Management System (CMS), and attended a two day training class on the Project Coordination Board (PCB) and Communications Manager (CM) at CERC. The June release of the CMS will be a stand-alone version running on UNIX platforms. The September release of the system will be integrated with the PCB/CM. The PCB and CM software will be provided to us from CERC as soon as possible so that we can perform our own internal evaluation. Draft copies of the PCB User Manual and three CM Programmer's Manuals were provided at the training class.

The purpose of the CMS is to provide mechanisms to effectively represent, manage and satisfy the constraints imposed on a product and its development. This is done through a collection of modules. One module allows for the creation and modification of constraints. Another module is used to manage the constraints, maintain consistency of the constraint network, evaluate constraints, and propagate variable values when necessary. Additional modules include a constraint propagation planner and a module to perform interval mathematics which is linked to Mathematica™. The possible forms that constraints could take include algebraic (*equality/inequality and linear/nonlinear*), numeric (*table look-up*), conditional (*if-then*), logical (*"X and Y"*), bidirectional (i.e. *x^3/(e^2.3) = 3.8y*), and unidirectional (*blackbox*). The types of constraints that the the first pilot site release will be able to handle are the bidirectional constraint represented by algebraic equations (linked to Mathematica™) and the blackbox constraint (unidirectional) which receive input and produce output through the evaluation of an external analysis program. The first release of the CMS will not have a user-friendly windowing interface, it will be command line driven. Many of our constraints for the MO system come in the form of conditional and logical constraints. For example, a constraint for the sheet metal panel/chassis environment could be "IF single_diameter_hole AND material_thickness = 0.125 AND maximum_bend_radius = 0.125 AND minimum_distance_to_edge_of_bend < 0.348 THEN punch ELSE drill_at_assembly". If we plan to integrate the first release of the CMS, we would have to utilize the blackbox constraint by writing our own external analysis program that would take our constraint format as input and return the result as output.

As explained earlier, the PCB is a system being developed to provide support for the coordination of the product development activities in a cooperative environment. The PCB provides common visibility and change notification through the common workspace (*cw*), planning and scheduling of activities through the task structure, monitoring progress of product development through the product structure, and computer support for team structure through messages. The PCB is composed of two modules: the *cw* (Common Workspace) module and the *da* (Design Agent) module. The *cw* module provides the functionality for project coordination, and the *da* module is an interface provided to product developers so they can interact with the *cw* module. The CM Communication Services (CS), the CM Directory Services (DS), and the CM Application Management Services (AMS) must be up and running for the PCB to work. For a given *cw* running there can be zero or more *da*'s running at a given point in time. It is also possible to have more than one *cw* running at a time. Currently, the PCB expects that you will plan your activities (project plan) in MacProject II and import that task structure into the PCB. The user can view any task or work order that appears in their network, add a task to the existing network, can acknowledge receiving a task, and indicate completion of their task. The PCB will automatically dispatch a task as previous tasks are completed, as well as, the Project Lead (user(s) with special privileges) can dispatch a task. The product structure can be loaded through a file or by using the Knowledge Server (i.e. importing from a ROSE database). Once the product structure is loaded, it is displayed using a hierarchical browser. The PCB allows the user to view (objects, constraints, and assertions), edit (object, attribute, and value), assert (values, constraints, and profiles), and save, load, or clear the knowledge base (*kb*). When the users are logged out of the PCB, messages concerning the release of new product models, constraint violations on attributes on which users have set a perspective, and any tasks that have been assigned are maintained for them and can be viewed at their convenience.

Since the PCB displays the product structure in a hierarchical fashion, Raytheon considered using the PCB as the front-end for the MO data model entry. The limitation of this PCB function is that it can import information into the PCB using the Knowledge Server, but the only output format is its *kb* file. We were hoping that it would not only be able to import the product structure from the ROSE database, but also to export the product structure to ROSE since our current plan is to store all the MO data in ROSE. They said that there are plans to provide this type of link because it has been requested by other pilot sites, but they could not provide us with a delivery date. We will request a copy of the Knowledge Server from CERC so that we can test the ROSE to PCB link.

The CM is a collection of modules that facilitates distributed computing in a heterogeneous network. It promotes the notion of a virtual network of resources which the project team members can exploit without any prior knowledge of the underlying physical network. The CM is useful for those that who are building transparent tools, virtual project networks, have access to remote tools, perform network tasks, perform message passing, and/or perform inter-process file transfers. The general functions of the CM include five modules: the session, network resource service (DS), application management service (AMS), task management service (TMS), and the communication service (CS). The session provides transparent and robust communication for programmers between a server and a client which aids in the building of transparent tools. The network resource service (DS) provides a database of network information for the end users, programmers, and CM administrator through the use of the *CM directory services* (Virtual Project Network). The application management service (AMS) allows remote execution of applications for programmers and CM administrators. The task management service *(TMS)* provides scheduling and distributing of jobs on a network for end users by creating a task file, comp; ing a task file, and executing a task file. The communication service (CS) is a communication package for inter-process file transfers for programmers.

Table 4-1. Summary of the CMS Evaluation Results

System Under Evaluation	MO Applicability	Maturity/Availability	Conclusion/ Recommendation
CMS	Model Process Model Rules as a set of Constraints.	Immature. Not available for pilot site installation until June.	Install and evaluate in June.
PCB	Scheduled for integration with the CMS by the end of September.	Immature. Attended training session, and scheduled for software installation as soon as possible.	Install and evaluate as soon as possible.
CM	Currently integrated with the PCB, and will be required when PCB/CM is integrated with the CMS at the end of September.	Immature. Attended training session, and scheduled for software installation as soon as possible.	Install and evaluate as soon as possible.

5. RM - Requirements Manager

5.1 Methods, Assumptions, and Procedures

5.1.1 Overview

The Requirements Manager is a system designed to manage product requirements, specifications and corporate policies to support concurrent engineering. The system allows the users to define requirements for a project or incorporate standard requirements through pointers (file name). The system also tracks parties interested in specific requirements and provides notification capabilities based on the status of a requirement. Status updates could include modifications of a requirement, product design driven violations of a requirement, or satisfaction of a requirement.

5.1.2 Proposed Use In MO

The purpose of integrating the MO manufacturing guideline functionality into the RM is to give the "top level" product development team insight into manufacturing requirements apart from MO analyses.

It is common practice for a manufacturer to document manufacturability, or producibility, guidelines that delineate standard manufacturing practices and acceptable design parameters. The purpose of these guidelines is to communicate the capabilities of the manufacturing process to the product design community to ensure that new product designs are specified within manufacturing capabilities. The guidelines delineate quantitative and qualitative producibility issues.

One of the functions of MO is to provide evaluation of manufacturing guidelines. For each guideline entry there is a related recommendation. The guidelines can be evaluated separately, or triggered based on the process analysis module within the MO system. Unlike the process selection constraints, manufacturability guideline violations may not cause alternative selection. The result could be an operation cost increase, for instance, the need for non-standard tooling, a yield loss, or a less tangible impact. These guidelines will also be entered into the Requirements Manager so that they are available to the product design team along with the other requirements placed on the design.

5.1.3 Evaluation Plan

The latest version of the RM is Version 2.0. Although this version was installed at CERC, it is not supported. While at the DICE Phase IV Kick-off meeting, we planned to attend a demonstration of the RM V3.0.

Raytheon has maintained a dialogue with CIMFLEX Teknowledge and has expressed its interest in being a beta test site.

5.2 Results and Discussion

The Requirements Manager (RM) is a system designed to manage product requirements, specifications and corporate policies to support concurrent engineering. Within the MO program, Raytheon planned to use the RM for manufacturability/producibility guidelines. Version 2.0 of the RM was installed at CERC, but is not supported. Raytheon decided to wait for the upcoming release, Version 3.0, and not to install V2.0. Raytheon attended a demonstration of the RM V3.0 at the DICE Phase IV Kick-off meeting. According to announced plans, the RM will be available for beta test at the end of March and released for production use at the end of June. Raytheon has maintained a dialogue with CIMFLEX and has expressed its interest in being a beta test site. The purpose of integrating the manufacturing guideline functionality into the RM is to give the "top level" product development team insight into manufacturing requirements apart from MO analyses. Raytheon will continue to develop the MO guideline functionality and will evaluate the RM when it is available. If the RM is not used, the implemented MO guideline functionality will include mechanisms such that it can be integrated into the product team environment as a module.

Table 5-1. Summary of the RM Evaluation Results

System Under Evaluation	MO Applicability	Maturity/Availability	Conclusion/ Recommendation
RM	Manage manufacturability/ producibility guidelines.	Immature. RM V3.0 not available for pilot installation.	Install and evaluate RM V3.0 when available.

6. Conclusions

The evaluation plan for each of the DICE tools considered for incorporation into the MO system environment, ROSE DBMS, CMS, PCB, CM, and RM, was performed. Table 6-1 contains the summary of the evaluation results.

After installing the Step Tool Kit (ROSE DBMS) and performing the eight tutorial lessons provided in the ROSE Tutorial Manual (reference 6), we felt that we had obtained an understanding of the basic ROSE functionality. We continued by developing a set of test cases to demonstrate the applicability of the ROSE DBMS to adequately model manufacturing process models and analysis data. This involved writing an EXPRESS schema to represent a sample of the process model data, generating the C++ classes with the use of the "express2c++" tool, and writing, compiling and executing various ROSE Applications (C++ programs). The developers learned the basics of the ROSE DBMS with the use of the reference and tutorial manuals provided with the software (References 4 and 6). To learn the more advanced features of ROSE, Raytheon plans on attending one of the training sessions conducted by STEP Tools. Raytheon believes that ROSE is quite stable. We encountered no difficulties during the installation or evaluation procedures. Based on the results of our sample test case, we believe that ROSE is able to handle the various types of manufacturing process model and analysis data required in the MO environment so we are planning on continuing the integration efforts with the ROSE System.

Since there was no pilot site version of the CMS software available, Raytheon visited CERC to received a demonstration of the current version of the Constraint Management System (CMS). The purpose of the CMS is to provide mechanisms to effectively represent, manage and satisfy the constraints imposed on a product and its development. The possible forms that a Constraints Manager would need include algebraic (*equality/inequality and linear/nonlinear*), numeric (*table look-up*), conditional (*if-then*), logical (*"X and Y"*), bidirectional (i.e. *x^3/(e^2.3) = 3.8y*), and unidirectional (*blackbox*). The types of constraints that the the first pilot site release will be able to handle are the bidirectional constraint represented by algebraic equations (linked to Mathematica™), and the blackbox constraint (unidirectional) which receive input and produce output through the evaluation of an external analysis program. Many of our constraints for the MO system come in the form of conditional and logical constraints. For example, a constraint for the sheet metal panel/chassis environment could be "IF single_diameter_hole AND material_thickness = 0.125 AND maximum_bend_radius = 0.125 AND minimum_distance_to_edge_of_bend < 0.348 THEN punch ELSE drill_at_assembly". If after the evaluation of the first release of the CMS we continue the integration efforts, we will have to utilize the blackbox constraint by writing our own external analysis program that would take our constraint format as input and return the result as output.

Since the CMS is scheduled to be integrated with the Project Coordination Board (PCB) and Communications Manager (CM), Raytheon attended a two day training class on the PCB and CM at CERC. The PCB is a system being developed to provide support for the coordination of the product development activities in a cooperative environment, and the CM is a collection of modules that facilitates distributed computing in a heterogeneous network; thereby, promoting the notion of a virtual network of resources which the project team members can exploit without any prior knowledge of the underlying physical network. Since the PCB displays the product structure in a hierarchical fashion, Raytheon considered using the PCB as the front-end for the MO data model entry. The limitation of this PCB function is that it can import information into the PCB using the Knowledge Server, but the only output format is its knowledge base (*kb*) file. We were hoping that it would not only be able to import the product structure from the ROSE database, but also to export the product structure to ROSE since our current plan is to store all the MO data in ROSE. The CERC pilot site support leader said that there is plans to provide this type of link because it has been requested by other pilot project sites, but they could not provide us with a delivery date. Since the Knowledge Server is the software that is required to translate data from the ROSE database into the PCB database, we will be requesting a copy of the Knowledge Server from CERC so that we can test the ROSE to PCB link. When Raytheon gets a copy of the pilot site version of the PCB/CM software, we will perform our own internal evaluation which will provide us with a basis for whether or not the systems could be of use in the MO environment. We plan to continue with the developed approach to this constraint manager function, and do not plan to incorporate the CMS, PCB, or CM in the MO design until after careful evaluation of each of the released systems.

The Requirements Manager (RM) is a system designed to manage product requirements, specifications and corporate policies to support concurrent engineering. Within the MO program, Raytheon planned to use the RM for manufacturability/producibility guidelines. Version 2.0 of the RM was installed at CERC, but is not supported. Raytheon decided to wait for the upcoming release, Version 3.0, and not to install V2.0. Raytheon will continue to develop the MO guideline functionality and will evaluate the RM when it is available. If the RM is not used, the implemented MO guideline functionality will include mechanisms such that it can be integrated into the product team environment as a module.

Table 6-1. Summary of the Evaluation Results

System Under Evaluation	MO Applicability	Maturity/Availability	Conclusion/ Recommendation
ROSE	Model, Store, and Manage Process Model and Analysis Data.	Stable. No difficulty with installation or evaluation procedures.	Plan on continuing integration efforts.
EDEX Editor	Code and edit MO Express Schemas.	No evaluation planned or conducted.	Plan to install.
CMS	Model Process Model Rules as a set of Constraints.	Immature. Not available for pilot site installation until June.	Install and evaluate in June.
PCB	Scheduled for integration with the CMS by the end of September.	Immature. Attended training session, and scheduled for software installation as soon as possible.	Install and evaluate as soon as possible.
CM	Currently integrated with the PCB, and will be required when PCB/CM is integrated with the CMS at the end of September.	Immature. Attended training session, and scheduled for software installation as soon as possible.	Install and evaluate as soon as possible.
RM	Manage manufacturability/ producibility guidelines.	Immature. RM V3.0 not available for pilot installation.	Install and evaluate RM V3.0 when available.

7. Recommendations

Raytheon has evaluated the ROSE system, the Constraint Management System (CMS), and the Requirements Manager, for product maturity and incorporation into the MO program. The evaluation of these tools was expanded to encompass the Project Coordination Board (PCB) and the Communications Manager (CM) because they are closely coupled to the CMS. Raytheon recommends using the ROSE system for MO implementation. At present, the CMS and RM are to immature to commit to for use in MO. The planned release for both the RM and CMS are during the early stage of the MO design cycle. Therefore, Raytheon recommends that both tools are tracked and further evaluation conducted when these tools are released. A final determination will be made at that time.

Raytheon proposed to use ROSE to store and manage the process models and analysis data for the MO system. As part of the evaluation process of ROSE, we developed a set of test cases to demonstrate the applicability of the ROSE DBMS to adequately model the manufacturing process models and analysis data. Based on the results of our sample test case, we believe that the ROSE DBMS is able to handle the various types of manufacturing process model and analysis data required in the MO environment. We recommended the continued use of the ROSE DBMS as the repository for all the process data files for the manufacturing processes and operations, as well as, the various analysis results for the MO system. To better understand the more advanced features of ROSE, attending one of the three day sessions that Step Tools conducts would be of great assistance in our future MO development efforts.

Raytheon proposed to use the CMS to provide a mechanism for constraining the product design to process capabilities. Since the pilot site version of the CMS software was unavailable for testing, Raytheon planned a trip to CERC for a demonstration of the current version of the CMS and a meeting with the developers. Based on the CERC visit, it was determined that the future versions of CMS will support the product/process constraint mechanism proposed in MO. Raytheon plans on installing the June release of the CMS. Raytheon will continue to develop the process selection functionality and will evaluate the CMS when it is available. If the CMS is not used, the implemented MO process selection functionality will include mechanisms such that it can be integrated into the product team environment as a module.

The types of constraints that the next CMS version will be able to handle are the bidirectional constraint represented by algebraic equations linked to Mathematica™, and the blackbox constraint (unidirectional) which receive input and produce output through the evaluation of an external analysis program. Many of the constraints that the MO system would require come in the form of conditional and logical constraints. In order to integrate the first release of the CMS, we would have to utilize the blackbox constraint by writing our own external analysis program that would take our constraint format as input and return the result as output.

Raytheon reviewed the PCB and CM which will be integrated with the September version of the CMS software. Based on the results of the hands on PCB/CM training class, Raytheon saw a potential to use the tools as a front-end for MO data model entry due to its hierarchical product structure display. We have reservations about the maturity of the system, and its interfaces with other systems (i.e. ROSE). The recommended plan is to get the PCB/CM software installed at Raytheon so that we can perform an internal evaluation using sample MO data. During our evaluation, we would be able to determine if the PCB/CM would add any value to the MO environment as an extension to the CMS or as a front-end to the MO process data model entry.

The Requirements Manager (RM) is a system designed to manage product requirements, specifications and corporate policies to support concurrent engineering. Within the MO program, Raytheon planned to use the RM for manufacturability/producibility guidelines. Version 2.0 of the RM was installed at CERC, but is not supported. Raytheon decided to wait for the upcoming release, Version 3.0, and not to install V2.0. Raytheon will continue to develop the MO guideline functionality and will evaluate the RM when it is available. If the RM is not used, the implemented MO guideline functionality will include mechanisms such that it can be integrated into the product team environment as a module.

8. References

1. BR-20558-1, 14 June 1991, DARPA Initiative In Concurrent Engineering (DICE) Manufacturing Optimization - Volume I - Technical.

2. CDRL No. 0002AC-1, 19 March 1992, Operational Concept Document For The Manufacturing Optimization (MO) System, Contract No. MDA972-92-C-0020.

3. Managing Constraints in Integrated and Cooperative Product Development, SAE Aerospace Technology Conference and Exposition, SAE Publication SP-886, Paper No. 912211, September 23–26, 1991, Long Beach, CA.

4. Reference Manual for the ROSE++ Data Manager, STEP Tools Inc.

5. Representing and Managing Constraints for Computer-Based Cooperative Product Development, Proceedings of the Third Annual National Symposium on Concurrent Engineering, Washington, D.C., June 10–14, 1991, pp. 473–504.

6. Tutorial Manual for the ROSE++ Data Manager, STEP Tools Inc.

7. User Manual for the Project Coordination Board of DICE, February 26, 1992, Contract No. MDA972-88-C-0047, CERC.

8. Notes

8.1 Acronyms

AMS	Application Management Service
CAEO	Computer Aided Engineering Operations
CDRL	Contract Data Requirement List
CM	Communications Manager
CMS	Constraint Management System
CS	Communication Service
DARPA	Defense Advanced Research Projects Agency
DBMS	Database Management System
DFMA	Design for Manufacturing and Assembly
DICE	DARPA Initiative In Concurrent Engineering
DS	Directory Service
EDEX	Editor for EXPRESS
GE	General Electric
MO	Manufacturing Optimization
PCB	Project Coordination Board
RM	Requirements Manager
ROSE	Rensselaer Object System For Engineering
RPI	Rensselaer Polytechnic Institute
TMS	Task Management Service

Appendix I - ROSE Evaluation Data

10. Process Model EXPRESS Schema

```
SCHEMA process_model;

   ENTITY Decision;
      name: STRING;
      parent: STRING;
      rules: LIST [0:?] OF STRING;
      END_ENTITY;

   ENTITY Operation
      SUBTYPE OF (Decision);
      op_desc: STRING;
      bid_code: INTEGER;
      END_ENTITY;

   ENTITY Step
      SUBTYPE OF (Decision);
      step_no: INTEGER;
      step_desc: STRING;
      cutting_tool: STRING;
      no_passes: INTEGER;
      step_time: REAL;
      END_ENTITY;

   ENTITY Process;
      sequence: LIST [0:?] OF Decision;
      END_ENTITY;

END_SCHEMA;
```

20. Express2c++ Generated Classes

20.1 process_model.h

```
#ifndef  process_model_h
#define  process_model_h

#include "Decision.h"
#include "Operation.h"
#include "Step.h"
#include "ListOfDecision.h"
#include "Process.h"
#endif
```

20.2 Decision.h

```
#ifndef  Decision_h
#define  Decision_h

#include "rose.h"
#define DecisionOffsets(subClass) \
      RoseStructureOffsets(subClass) \
      ROSE_SUPERCLASS_OFFSET(subClass,Decision)


ROSE_DECLARE (Decision) : virtual public RoseStructure {

 private:
      STR PERSISTENT_name;
      STR PERSISTENT_parent;
      ListOfString *  PERSISTENT_rules;

 public:
      ROSE_DECLARE_MEMBERS(Decision);

/* Access and Update Methods */

/* name Access Methods */
STR  name()
{     return ROSE_GET_PRIM (STR,PERSISTENT_name);
}
Decision * name (STR  aname)
{     ROSE_PUT_PRIM (STR,PERSISTENT_name,aname);
      return this;
}

/* parent Access Methods */
STR  parent()
{     return ROSE_GET_PRIM (STR,PERSISTENT_parent);
}
Decision * parent (STR  aparent)
{     ROSE_PUT_PRIM (STR,PERSISTENT_parent,aparent);
      return this;
}

/* rules Access Methods */
ListOfString * rules();
Decision * rules (ListOfString *  arules)
{     ROSE_PUT_OBJ (ListOfString,PERSISTENT_rules,arules);
      return this;
}
```

```
/* Default Constructor - This constructor may be modified,
 *  but *DO NOT* add any calls that would initialize ROSE.
 */
Decision () {
      PERSISTENT_name = NULL;
      PERSISTENT_parent = NULL;
      PERSISTENT_rules = NULL;
}

Decision  (
      STR aname,
      STR aparent,
      ListOfString *  arules );
};
#endif
```

20.3 Decision.c

```
#ifndef  Decision_c
#define  Decision_c


// Class Decision
#include "Decision.h"
ROSE_BEGIN_MEMBERS("Decision",Decision,DecisionOffsets)
      ROSE_SCHEMA_NAME("process_model")
      ROSE_VIRTUALSUPERCLASS (RoseStructure)
      ROSE_VARIABLE(STR,PERSISTENT_name,"name")
      ROSE_VARIABLE(STR,PERSISTENT_parent,"parent")
      ROSE_VARIABLE(ListOfString,PERSISTENT_rules,"rules")
ROSE_END_MEMBERS;

ListOfString * Decision :: rules()
{     if( !PERSISTENT_rules)
      if( this->isPersistent())
            rules (pnewIn (design()) ListOfString);
      else  rules (new ListOfString);
      return ROSE_GET_OBJ (ListOfString,PERSISTENT_rules);
}

/* Custom Constructor */
Decision::Decision (
      STR aname,
      STR aparent,
      ListOfString *  arules )
{
      name (aname);
      parent (aparent);
      rules (arules);
 }
#endif
```

20.4 Operation.h

```
#ifndef  Operation_h
#define  Operation_h

#include "rose.h"
#include "Decision.h"
#define OperationOffsets(subClass) \
      DecisionOffsets(subClass) \
```

```
      ROSE_SUPERCLASS_OFFSET(subClass,Operation)

ROSE_DECLARE (Operation) : virtual public Decision {
 private:
      STR PERSISTENT_op_desc;
      int PERSISTENT_bid_code;

 public:
      ROSE_DECLARE_MEMBERS(Operation);

/* Access and Update Methods */

/* op_desc Access Methods */
STR  op_desc()
{     return ROSE_GET_PRIM (STR,PERSISTENT_op_desc);
}
Operation * op_desc (STR  aop_desc)
{     ROSE_PUT_PRIM (STR,PERSISTENT_op_desc,aop_desc);
      return this;
}

/* bid_code Access Methods */
int  bid_code()
{     return ROSE_GET_PRIM (int,PERSISTENT_bid_code);
}
Operation * bid_code (int  abid_code)
{     ROSE_PUT_PRIM (int,PERSISTENT_bid_code,abid_code);
      return this;
}

/* Default Constructor - This constructor may be modified,
 *  but *DO NOT* add any calls that would initialize ROSE.
 */
Operation () {
      PERSISTENT_op_desc = NULL;
      PERSISTENT_bid_code = NULL;
}

Operation  (
      STR aname,
      STR aparent,
      ListOfString *  arules,
      STR aop_desc,
      int abid_code );
};
#endif
```

20.5 Operation.c

```
#ifndef  Operation_c
#define  Operation_c


// Class Operation
#include "Operation.h"
ROSE_BEGIN_MEMBERS("Operation",Operation,OperationOffsets)
      ROSE_SCHEMA_NAME("process_model")
      ROSE_VIRTUALSUPERCLASS (Decision)
      ROSE_VARIABLE(STR,PERSISTENT_op_desc,"op_desc")
      ROSE_VARIABLE(int,PERSISTENT_bid_code,"bid_code")
ROSE_END_MEMBERS;
```

```
/* Custom Constructor */
Operation::Operation (
      STR aname,
      STR aparent,
      ListOfString *  arules,
      STR aop_desc,
      int abid_code )
{
      name (aname);
      parent (aparent);
      rules (arules);
      op_desc (aop_desc);
      bid_code (abid_code);
 }
#endif
```

20.6 Step.h

```
#ifndef  Step_h
#define  Step_h

#include "rose.h"
#include "Decision.h"
#define StepOffsets(subClass) \
      DecisionOffsets(subClass) \
      ROSE_SUPERCLASS_OFFSET(subClass,Step)


ROSE_DECLARE (Step) : virtual public Decision {
 private:
      int PERSISTENT_step_no;
      STR PERSISTENT_step_desc;
      STR PERSISTENT_cutting_tool;
      int PERSISTENT_no_passes;
      float PERSISTENT_step_time;

 public:
      ROSE_DECLARE_MEMBERS(Step);

/* Access and Update Methods */

/* step_no Access Methods */
int  step_no()
{     return ROSE_GET_PRIM (int,PERSISTENT_step_no);
}
Step * step_no (int  astep_no)
{     ROSE_PUT_PRIM (int,PERSISTENT_step_no,astep_no);
      return this;
}

/* step_desc Access Methods */
STR  step_desc()
{     return ROSE_GET_PRIM (STR,PERSISTENT_step_desc);
}
Step * step_desc (STR  astep_desc)
{     ROSE_PUT_PRIM (STR,PERSISTENT_step_desc,astep_desc);
      return this;
}

/* cutting_tool Access Methods */
STR  cutting_tool()
{     return ROSE_GET_PRIM (STR,PERSISTENT_cutting_tool);
}
```

```
Step * cutting_tool (STR  acutting_tool)
{      ROSE_PUT_PRIM (STR,PERSISTENT_cutting_tool,acutting_tool);
       return this;
}

/* no_passes Access Methods */
int  no_passes()
{      return ROSE_GET_PRIM (int,PERSISTENT_no_passes);
}
Step * no_passes (int  ano_passes)
{      ROSE_PUT_PRIM (int,PERSISTENT_no_passes,ano_passes);
       return this;
}

/* step_time Access Methods */
float  step_time()
{      return ROSE_GET_PRIM (float,PERSISTENT_step_time);
}
Step * step_time (float  astep_time)
{      ROSE_PUT_PRIM (float,PERSISTENT_step_time,astep_time);
       return this;
}

/* Default Constructor - This constructor may be modified,
 *  but *DO NOT* add any calls that would initialize ROSE.
 */
Step () {
       PERSISTENT_step_no = NULL;
       PERSISTENT_step_desc = NULL;
       PERSISTENT_cutting_tool = NULL;
       PERSISTENT_no_passes = NULL;
       PERSISTENT_step_time = NULL;
}

Step  (
       STR aname,
       STR aparent,
       ListOfString *  arules,
       int astep_no,
       STR astep_desc,
       STR acutting_tool,
       int ano_passes,
       float astep_time );
};
#endif
```

20.7 Step.c

```
#ifndef  Step_c
#define  Step_c


// Class Step
#include "Step.h"
ROSE_BEGIN_MEMBERS("Step",Step,StepOffsets)
       ROSE_SCHEMA_NAME("process_model")
       ROSE_VIRTUALSUPERCLASS (Decision)
       ROSE_VARIABLE(int,PERSISTENT_step_no,"step_no")
       ROSE_VARIABLE(STR,PERSISTENT_step_desc,"step_desc")
       ROSE_VARIABLE(STR,PERSISTENT_cutting_tool,"cutting_tool")
       ROSE_VARIABLE(int,PERSISTENT_no_passes,"no_passes")
       ROSE_VARIABLE(float,PERSISTENT_step_time,"step_time")
ROSE_END_MEMBERS;
```

```
/* Custom Constructor */
Step::Step (
      STR aname,
      STR aparent,
      ListOfString *  arules,
      int astep_no,
      STR astep_desc,
      STR acutting_tool,
      int ano_passes,
      float astep_time )
{
      name (aname);
      parent (aparent);
      rules (arules);
      step_no (astep_no);
      step_desc (astep_desc);
      cutting_tool (acutting_tool);
      no_passes (ano_passes);
      step_time (astep_time);
 }
#endif
```

20.8 ListOfDecision.h

```
#ifndef  ListOfDecision_h
#define  ListOfDecision_h


// Class ListOfDecision
#include "rose.h"
ROSE_DECLARE (Decision);
declare(List,Decision);
#endif
```

20.9 ListOfDecision.c

```
#ifndef  ListOfDecision_c
#define  ListOfDecision_c

#include "ListOfDecision.h"

implement2(List,Decision,"process_model");
#endif
```

20.10 Process.h

```
#ifndef  Process_h
#define  Process_h

#include "rose.h"
ROSE_DECLARE (ListOfDecision);
#define ProcessOffsets(subClass) \
      RoseStructureOffsets(subClass) \
      ROSE_SUPERCLASS_OFFSET(subClass,Process)


ROSE_DECLARE (Process) : virtual public RoseStructure {

 private:
```

```
      ListOfDecision *  PERSISTENT_sequence;

 public:
      ROSE_DECLARE_MEMBERS(Process);

/* Access and Update Methods */

/* sequence Access Methods */
ListOfDecision * sequence();
Process * sequence (ListOfDecision *  asequence)
{     ROSE_PUT_OBJ (ListOfDecision,PERSISTENT_sequence,asequence);
      return this;
}

/* Default Constructor - This constructor may be modified,
 *  but *DO NOT* add any calls that would initialize ROSE.
 */
Process () {
      PERSISTENT_sequence = NULL;
}

Process   (
      ListOfDecision *  asequence );
};
#endif
```

20.11 Process.c

```
#ifndef  Process_c
#define  Process_c


// Class Process
#include "Process.h"
#include "ListOfDecision.h"
ROSE_BEGIN_MEMBERS("Process",Process,ProcessOffsets)
      ROSE_SCHEMA_NAME("process_model")
      ROSE_VIRTUALSUPERCLASS (RoseStructure)
      ROSE_VARIABLE(ListOfDecision,PERSISTENT_sequence,"sequence")
ROSE_END_MEMBERS;

ListOfDecision * Process :: sequence()
{     if( !PERSISTENT_sequence)
      if( this->isPersistent())
            sequence (pnewIn (design()) ListOfDecision);
      else  sequence (new ListOfDecision);
      return ROSE_GET_OBJ (ListOfDecision,PERSISTENT_sequence);
}

/* Custom Constructor */
Process::Process (
      ListOfDecision *  asequence )
{
      sequence (asequence);
 }
#endif
```

30. Compiled Process Model Schema

```
Format = "rose_r3.0"

ROSE_OIDS (
	(0 = 0x000000000000000000000000000000000000000)
	(1 = 0x007D210704000029B6822F00001C7E0300000000)
)

ROSE_DESIGN (RoseDesign
	name: "process_model"
	root: $
	keyword_table: $
	name_table: <1-1>
	schemas: (<1-30> ListOfRoseDesign
		<"keystone3_0">)
)

STEP_OBJECTS (
	(<1-1> DictionaryOfRoseObject
		listOfKeys: (<1-2> ListOfSTR
			"Decision"
			"Operation"
			"Step"
			"Process"
			"ListOfDecision")
		listOfValues: (<1-3> ListOfRoseObject
			(<1-0> RoseDomain
				name: "Decision"
				listOfSuper: (<1-8> ListOfRoseDomain
					<"keystone3_0" 0-600>)
				listOfAttribute: (<1-9> ListOfRoseAttribute
					(<1-10> RoseAttribute
						name: "name"
						domain: <"keystone3_0" 0-4>)
					(<1-11> RoseAttribute
						name: "parent"
						domain: <"keystone3_0" 0-4>)
					(<1-12> RoseAttribute
						name: "rules"
						domain: <"keystone3_0" 0-1180>)))
			(<1-4> RoseDomain
				name: "Operation"
				listOfSuper: (<1-13> ListOfRoseDomain
					<1-0>)
				listOfAttribute: (<1-14> ListOfRoseAttribute
					(<1-15> RoseAttribute
						name: "op_desc"
						domain: <"keystone3_0" 0-4>)
					(<1-16> RoseAttribute
						name: "bid_code"
						domain: <"keystone3_0" 0-1>)))
			(<1-5> RoseDomain
				name: "Step"
				listOfSuper: (<1-17> ListOfRoseDomain
					<1-0>)
				listOfAttribute: (<1-18> ListOfRoseAttribute
					(<1-19> RoseAttribute
						name: "step_no"
						domain: <"keystone3_0" 0-1>)
					(<1-20> RoseAttribute
						name: "step_desc"
						domain: <"keystone3_0" 0-4>)
```

```
                (<1-21> RoseAttribute
                      name: "cutting_tool"
                      domain: <"keystone3_0" 0-4>)
                (<1-22> RoseAttribute
                      name: "no_passes"
                      domain: <"keystone3_0" 0-1>)
                (<1-23> RoseAttribute
                      name: "step_time"
                      domain: <"keystone3_0" 0-2>)))
(<1-6> RoseDomain
      name: "Process"
      listOfSuper: (<1-24> ListOfRoseDomain
            <"keystone3_0" 0-600>)
      listOfAttribute: (<1-25> ListOfRoseAttribute
            (<1-26> RoseAttribute
                  name: "sequence"
                  domain: (<1-7> RoseDomain
                    name: "ListOfDecision"
                    listOfSuper: (<1-27> ListOfRoseDomain
                         <"keystone3_0" 0-1102>)
                    listOfAttribute: (<1-28> ListOfRoseAttribute
                         (<1-29> RoseAttribute
                                 name: "Decision"
                                 domain: <1-0>))))))
<1-7>))

)
```

40. C++ Program - Instantiates Process Model Objects from ASCII Process Data File

```
/* Include headers for ROSE Library and program classes */
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <fcntl.h>

#include "rose.h"
#include "process_model.h"

#define MAXLINE 81
#define MAXBUF  256

void strip_trailing_blanks(char *);

main()
{
   char          a_fn[MAXLINE];
   char          buff[MAXBUF];
   FILE          *inp;
   ListOfString *lrules;
   float         fl;
   int           num_rules, x;


   printf("  ENTER INPUT ASCII DATA FILENAME: ");
   scanf("%s",a_fn);

   /*** Open Input ASCII Process Data File ***/
   if ((inp = fopen(a_fn, "r")) == (FILE *) NULL)
   {
      printf("ERROR opening %s DATA FILE\n", a_fn);
      exit(1);
   }

   /*** Create a rose database to store process_model type objects ***/
   ROSE.newDesign ("process_data");

   ListOfDecision *xyz = pnew ListOfDecision;

   while (fgets(buff, MAXBUF, inp) != NULL)
   {
      printf("'%s'", buff);

      if (strncmp(buff, "#", 1) == 0)   /* comment */
        continue;

      else if (strncmp(buff, "Decision", 8) == 0) /* Decision Is Being Added */
      {
         printf("Inside Decision\n\n");
         /*** Create Decision Objects in the database ***/
         Decision *d1 = pnew Decision;

         if (fgets(buff,MAXBUF,inp) != NULL)
         {
           strip_trailing_blanks(buff);
           printf("'%s'", buff);
           d1->name(buff);
         }
         if (fgets(buff,MAXBUF,inp) != NULL)
```

```
        {
          strip_trailing_blanks(buff);
          printf("'%s'", buff);
          dl->parent(buff);
        }

        lrules = pnew ListOfString;
        if (fgets(buff,MAXBUF,inp) != NULL)
        {
          strip_trailing_blanks(buff);
          printf("'%s'", buff);
          sscanf(buff, "%d", &num_rules);
        }
        for (int i=0; i<num_rules; i++)
        {
           if (fgets(buff,MAXBUF,inp) != NULL)
           {
             strip_trailing_blanks(buff);
             printf("'%s'", buff);
             lrules->add(buff);
           }
        }
        dl->rules(lrules);
        xyz->add(dl);
     }

     else if (strncmp(buff, "Operation", 9) == 0) /* Operation Is Being Added */
     {
        printf("Inside Operation\n\n");
        /*** Create Operation Objects in the database ***/
        Operation *oper1 = pnew Operation;

        if (fgets(buff,MAXBUF,inp) != NULL)
        {
          strip_trailing_blanks(buff);
          printf("'%s'", buff);
          oper1->name(buff);
        }
        if (fgets(buff,MAXBUF,inp) != NULL)
        {
          strip_trailing_blanks(buff);
          printf("'%s'", buff);
          oper1->parent(buff);
        }

        lrules = pnew ListOfString;
        if (fgets(buff,MAXBUF,inp) != NULL)
        {
          strip_trailing_blanks(buff);
          printf("'%s'", buff);
          sscanf(buff, "%d", &num_rules);
        }
        for (int i=0; i<num_rules; i++)
        {
           if (fgets(buff,MAXBUF,inp) != NULL)
           {
             strip_trailing_blanks(buff);
             printf("'%s'", buff);
             lrules->add(buff);
           }
        }
        oper1->rules(lrules);

        if (fgets(buff,MAXBUF,inp) != NULL)
```

```
      {
        strip_trailing_blanks(buff);
        printf("'%s'", buff);
        oper1->op_desc(buff);
      }
      if (fgets(buff,MAXBUF,inp) != NULL)
      {
        strip_trailing_blanks(buff);
        printf("'%s'", buff);
        sscanf(buff, "%d", &x);
        oper1->bid_code(x);
      }
      xyz->add(oper1);
   }

   else  /* Step Is Being Added */
   {
      printf("Inside Step\n\n");

      /*** Create Step Objects in the database ***/
      Step *step1 = pnew Step;

      if (fgets(buff,MAXBUF,inp) != NULL)
      {
        strip_trailing_blanks(buff);
        printf("'%s'", buff);
        step1->name(buff);
      }
      if (fgets(buff,MAXBUF,inp) != NULL)
      {
        strip_trailing_blanks(buff);
        printf("'%s'", buff);
        step1->parent(buff);
      }

      lrules = pnew ListOfString;
      if (fgets(buff,MAXBUF,inp) != NULL)
      {
        strip_trailing_blanks(buff);
        printf("'%s'", buff);
        sscanf(buff, "%d", &num_rules);
      }
      for (int i=0; i<num_rules; i++)
      {
          if (fgets(buff,MAXBUF,inp) != NULL)
          {
            strip_trailing_blanks(buff);
            printf("'%s'", buff);
            lrules->add(buff);
          }
      }
      step1->rules(lrules);

      if (fgets(buff,MAXBUF,inp) != NULL)
      {
        strip_trailing_blanks(buff);
        printf("'%s'", buff);
        sscanf(buff, "%d", &x);
        step1->step_no(x);
      }
      if (fgets(buff,MAXBUF,inp) != NULL)
      {
        strip_trailing_blanks(buff);
        printf("'%s'", buff);
```

```
          step1->step_desc(buff);
        }
        if (fgets(buff,MAXBUF,inp) != NULL)
        {
          strip_trailing_blanks(buff);
          printf("'%s'", buff);
          step1->cutting_tool(buff);
        }
        if (fgets(buff,MAXBUF,inp) != NULL)
        {
          strip_trailing_blanks(buff);
          printf("'%s'", buff);
          sscanf(buff, "%d", &x);
          step1->no_passes(x);
        }
        if (fgets(buff,MAXBUF,inp) != NULL)
        {
          strip_trailing_blanks(buff);
          printf("'%s'", buff);
          sscanf(buff, "%f", &fl);
          step1->step_time(fl);
        }
        xyz->add(step1);
      }

    }  /* end of while fgets */

    /*** Display all STEP Objects in the process data model ***/
    ROSE.display();

    /*** Save the process data model ***/
    ROSE.saveDesign();

    fclose(inp);
}



/*** Strips trailing blanks ***/
void strip_trailing_blanks(char *string)
{
  int len, k;

  len = strlen(string);    /* calculate length of string */
  string[len-1] = '\0';    /* delete newline character from end of line */

  /* get ride of trailing blanks */
  for (k=len-2; k>0;k--)
  {
    if (string[k] == ' ')
      string[k] = '\0';
    else break;
  }
}
```

50. Input ASCII Process Data File

```
Decision
Panel/Chassis
NULL
3
ALUM
STEEL
CRES
#
Decision
Sheet
Panel/Chassis
1
part_qty < 1000
#
Operation
Shear
Sheet
2
!hole
hole_qty < 8
Shear to Size
30
#
Decision
Punch
Sheet
2
single_d
double_d
#
Operation
Single Station Punch Press
Punch
1
part_qty < 25
Perform manual punching operation
35
#
Operation
CNC Punch
Punch
1
part_qty >= 25
Perform CNC punching operation
40
#
Operation
Holemaking
Sheet
3
single_diam
counterbore
countersunk
Holemaking in the flat
30
#
Step
Ream
Holemaking
1
always pick
```

```
1
ream hole
ream_tool
2
0.25
#
Step
Bore
Holemaking
1
counterbore
2
bore hole
bore_tool
1
0.15
#
Step
Drill
Holemaking
1
always pick
3
drill hole
drill_tool
3
0.45
#
Operation
TimeSaver
Sheet
1
always pick
Time Saving Operation in the flat
30
#
Decision
Strip
Panel/Chassis
1
part_qty >= 1000
```

60. Instantiated Process Model ROSE Output Database

```
Format = "rose_r3.0"

ROSE_OIDS (
      (0 = 0x0000000000000000000000000000000000000000)
      (1 = 0x007D210704000029B6836500001CE60200000000)
)

ROSE_DESIGN (RoseDesign
      name: "process_data"
      root: $
      keyword_table: $
      name_table: $
      schemas: (<1-25> ListOfRoseDesign
            <"keystone3_0">
            <"process_model">)
)

STEP_OBJECTS (
      (<1-0> ListOfDecision
            (<1-1> Decision
                  name: "Panel/Chassis"
                  parent: "NULL"
                  rules: (<1-2> ListOfSTR
                        "ALUM"
                        "STEEL"
                        "CRES"))
            (<1-3> Decision
                  name: "Sheet"
                  parent: "Panel/Chassis"
                  rules: (<1-4> ListOfSTR
                        "part_qty < 1000"))
            (<1-5> Operation
                  name: "Shear"
                  parent: "Sheet"
                  rules: (<1-6> ListOfSTR
                        "!hole"
                        "hole_qty < 8")
                  op_desc: "Shear to Size"
                  bid_code: 30)
            (<1-7> Decision
                  name: "Punch"
                  parent: "Sheet"
                  rules: (<1-8> ListOfSTR
                        "single_d"
                        "double_d"))
            (<1-9> Operation
                  name: "Single Station Punch Press"
                  parent: "Punch"
                  rules: (<1-10> ListOfSTR
                        "part_qty < 25")
                  op_desc: "Perform manual punching operation"
                  bid_code: 35)
            (<1-11> Operation
                  name: "CNC Punch"
                  parent: "Punch"
                  rules: (<1-12> ListOfSTR
                        "part_qty >= 25")
                  op_desc: "Perform CNC punching operation"
                  bid_code: 40)
            (<1-13> Operation
                  name: "Holemaking"
```

```
        parent: "Sheet"
        rules: (<1-14>     OfSTR
              "single_[illegible]
              "counterbore"
              "countersunk")
        op_desc: "Holemaking in the flat"
        bid_code: 30)
  (<1-15> Step
        name: "Ream"
        parent: "Holemaking"
        rules: (<1-16> ListOfSTR
              "always pick")
        step_no: 1
        step_desc: "ream hole"
        cutting_tool: "ream_tool"
        no_passes: 2
        step_time: 0.25)
  (<1-17> Step
        name: "Bore"
        parent: "Holemaking"
        rules: (<1-18> ListOfSTR
              "counterbore")
        step_no: 2
        step_desc: "bore hole"
        cutting_tool: "bore_tool"
        no_passes: 1
        step_time: 0.15)
  (<1-19> Step
        name: "Drill"
        parent: "Holemaking"
        rules: (<1-20> ListOfSTR
              "always pick")
        step_no: 3
        step_desc: "drill hole"
        cutting_tool: "drill_tool"
        no_passes: 3
        step_time: 0.45)
  (<1-21> Operation
        name: "TimeSaver"
        parent: "Sheet"
        rules: (<1-22> ListOfSTR
              "always pick")
        op_desc: "Time Saving Operation in the flat"
        bid_code: 30)
  (<1-23> Decision
        name: "Strip"
        parent: "Panel/Chassis"
        rules: (<1-24> ListOfSTR
              "part_qty >= 1000")))

)
```

70. C++ Program - Selects Two Process Plan Sequences

```
/* Include headers for ROSE Library and program classes */
#include <stdio.h>

#include "rose.h"
#include "process_model.h"


main()
{
   int x = 0;
   int x1 = 0;
   ListOfDecision dList;

   /*** Read in the Process Data File ***/
   ROSE.useDesign("process_data");
   ROSE.findObjects (&dList);

   ROSE.newDesign("selected");

   int n = dList.size();
   printf("N = %d\n", n);

   Process *p1 = pnew Process;
   ListOfDecision *n1list = pnew ListOfDecision;

   for (int i=0; i< n; i++)
   {
     printf("Name is '%s'\n", dList[i]->name());
     if (strcmp(dList[i]->name(), "Holemaking") == 0)
     {
       x++;
       n1list->add(dList[i]);
     }
     else if (strcmp(dList[i]->name(), "Ream") == 0)
     {
       x++;
       n1list->add(dList[i]);
     }
     else if (strcmp(dList[i]->name(), "Bore") == 0)
     {
       x++;
       n1list->add(dList[i]);
     }
     else if (strcmp(dList[i]->name(), "Drill") == 0)
     {
       x++;
       n1list->add(dList[i]);
     }
     p1->sequence(n1list);
   }


   Process *p2 = pnew Process;
   ListOfDecision *n2list = pnew ListOfDecision;

   for (i=0; i< n; i++)
   {
     printf("Name is '%s'\n", dList[i]->name());
     if (strcmp(dList[i]->name(), "Punch") == 0)
     {
       x1++;
```

```
      n2list->add(dList[i]);
    }
    else if (strcmp(dList[i]->name(), "Single Station Punch Press") == 0)
    {
      x1++;
      n2list->add(dList[i]);
    }
    else if (strcmp(dList[i]->name(), "CNC Punch") == 0)
    {
      x1++;
      n2list->add(dList[i]);
    }
    p2->sequence(n2list);
  }

  /*** Save the selected process ***/
  ROSE.saveDesign();
  ROSE.display();

  for (i=0; i < x; i++)
    printf("N1List is '%s'\n", (*n1list)[i]->name());

  for (i=0; i < x1; i++)
    printf("N2List is '%s'\n", (*n2list)[i]->name());
}
```

80. Output ROSE Database Containing Two Process Plans

```
Format = "rose_r3.0"

ROSE_OIDS (
	(1 = 0x0000000000000000000000000000000000000000)
	(2 = 0x007D210704000029B685B200001D610200000000)
	(0 = 0x007D210704000029B6836500001CE60200000000)
)

ROSE_DESIGN (RoseDesign
	name: "selected"
	root: $
	keyword_table: $
	name_table: $
	schemas: (<2-4> ListOfRoseDesign
		<"process_model">
		<"keystone3_0">)
)

STEP_OBJECTS (
	(<2-0> Process
		sequence: (<2-1> ListOfDecision
			<"process_data" 0-13>
			<"process_data" 0-15>
			<"process_data" 0-17>
			<"process_data" 0-19>))
	(<2-2> Process
		sequence: (<2-3> ListOfDecision
			<"process_data" 0-7>
			<"process_data" 0-9>
			<"process_data" 0-11>))

)
```

90. C++ Program - Read In Process Plans ROSE Database

```
/* Include headers for ROSE Library and program classes */
#include <stdio.h>

#include "rose.h"
#include "process_model.h"

declare(List,Process);
implement(List,Process);

main()
{
   int x = 0;
   int x1 = 0;
   ListOfProcess dList;
   ListOfDecision *seq;

   /*** Read in the Process Data File ***/
   ROSE.useDesign("selected");
   ROSE.findObjects (&dList);

   int n = dList.size();
   printf("N = %d\n", n);

   for (int i=0; i< n; i++)
   {
     seq = dList[i]->sequence();
     for (int j=0; j< seq->size(); j++)
       printf("Process%d = %s\n", i+1, (*seq)[j]->name());
   }

   ROSE.display();
}
```

Distribution List

DPRO-Raytheon
C/O Raytheon Company
Spencer Lab., Wayside Ave.
(one copy)

Defense Advanced Research Projects Agency
ATTN: Defense Sciences Office; Dr. H. Lee Buchanan
Virginia Square Plaza
3701 N. Fairfax Drive
Arlington, VA. 22203-1714
(one copy)

Defense Advanced Research Projects Agency
ATTN: Electronic Systems Technology Office; Capt. Nicholas J. Niclerio, USAF
Virginia Square Plaza
3701 N. Fairfax Drive
Arlington, VA. 22203-1714
(one copy)

Defense Advanced Research Projects Agency
ATTN: Contracts Management Office; Mr. Donald C. Sharkus
Virginia Square Plaza
3701 N. Fairfax Drive
Arlington, VA. 22203-1714
(one copy)

Defense Technical Information Center
Building 5, Cameron Station
ATTN: Selections
Alexandria, VA 22304
(two copies)